Towards a Framework for Hypermedia Scenarios

Reginald L. Hobbs
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332, USA
E-mail: reggie@cc.gatech.edu

Dr. Colin Potts
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332, USA
E-mail: potts@cc.gatech.edu

ABSTRACT
Scenarios have been used in many disciplines, such as software engineering, cognitive science, and HCI (Human-Computer Interaction) to aid in decision making, comprehension, design, and training. Scenarios are instances of behavior of the systems, or "stories" about how the system will or should look. These scenarios are used largely as points of discussion in each of these cases. Building new scenarios or analyzing existing scenarios orient the discussion in collaborative activities and increase understanding in single user tasks. How can a scenario structure be defined that will support the widest range of discussion/comprehension activities and remain content- and access-independent? Scenarios can be structured as a document. In some ways, it is similar to a screenplay document from the film industry. The use of a markup language will allow scenario documents to take advantage of hypermedia representations of the components. Building appropriate tools to interpret the Scenario Markup Language (SCML) will support the creation of different representations of scenarios (such as storyboards, audiovisual segments, narratives, etc.) from the same data. Interactions between SCML and tool functionality will also support scenario authoring, dependency analysis, and structured walkthroughs.

KEYWORDS: Scenarios, hypermedia, software engineering, SGML, collaboration

INTRODUCTION
Currently, software developers and programmers are using scenarios (usually in the form of rapid prototyping or computer simulations) to assist them in system development. Scenarios are being used throughout the design process to examine alternate methods and techniques.[10] These "what-if" studies allow them to refine the design and requirements of a system before (and sometimes during) implementation. By catching potential system errors and design problems early in system evolution, costly redesigns are minimized.

Scenarios are not only used in computer system development. Scenarios are becoming an integral part in the design and analysis of systems across many different fields of study. They occur in our day-to-day interactions, as well. Consider the following scenarios:

- A visitor from out-of-town stops to ask directions from Hartsfield Atlanta Airport to the campus at Georgia Tech. A knowlegeable resident explains how to get there by describing the navigation from the point-of-view of the visitor: "First you get onto the interstate going North, then take exit 100....".
- A software developer creates a prototype user interface for a word processing package and discusses it with a potential user by asking what type of tasks would be performed.
- An army commander has a tactical planning briefing with his staff where they discuss alternate ways of meeting mission objectives by creating a storyboard of tactics.
- A mathematical model is constructed to run a stock market simulation based on a particular set of variables. The outcome of the simulation is then used to predict market fluctuations and the factors that could cause them.

Each of these situations is an example of a scenario. A scenario is an instance of a system behavior, a sequence of events and interactions that describe a specific case of a systems' function. Whether it is for the purpose of comprehension (out-of-town visitor),design (user interface prototype), collaborative decision making (Army tactical planning briefing), or dependency analysis (stock market simulation), a scenario is a powerful method for discussing system behavior.

During a 1993 workshop on user-oriented design representations, several leading practitioners in the fields of Human-Computer Interaction, requirements engineering, computer-supported cooperative work (CSCW), and other disciplines within software engineering attempted to focus attention on the concept of scenario-based design.[5] The result was a collection of case studies, design perspectives, and approaches that illustrated the usefulness of scenarios to enhance traditional system design. Some of these studies on scenarios proposed methods for establishing categories of scenarios as design artifacts to be used during system development.[8] This effort outlined a list of roles of scenarios throughout during the

phases of system development[6]:

- Requirements Analysis
- User-Designer Communication
- Design Rationale
- Envisionment
- Software Design
- Implementation
- Documentation and Training
- Evaluation
- Abstraction
- Team Building

In each of these roles, scenarios are an informal approach to the refinement of software artifacts. The informality of scenarios is due to the fact that they can take on many forms, making specific format or standard unnecessary. Software prototyping, textual descriptions, storyboarding, user-interface mockups, simulations can all be considered forms of scenarios.

Some influential cognitive scientists have elevated narrative to a central position in their models of explanation and inference and in the application of these models to pedagogy and educational technology.[16] In some ways, scenarios are analagous to the cognitive science notions of mentals models and thought experiments. Actually, scenarios are closer to a composite of the terms. A mental model is an internal representation of the outside world. When asked to "picture the earth", most people have an internal visualization of a spinning, spherical object with patches of blue and brown representing oceans and land. A thought experiment is a hypothesis testing process where a person visualizes a sequence of events to determine a possible outcome. If the internal visualization is a sufficiently reasonable model of the environment under which the events occur, the outcome will illustrate possible solutions to the problem. A scenario incorporates describing these internal models (with different levels of detail) with a hypothesis-testing focus.

By separating the content and application of a scenario from its structure, an environment can be created that could support a wide range of these activities. The problem, is how to define this structure and develop tools to support the various scenario-driven activities? In this paper, we introduce the concept of a hypermedia scenario document, or **hyperscenario**. Building the scenario framework with hypermedia structures lets us create different presentation styles for the same scenario. Hyperlinked connections between text, graphics, and other multimedia allow multiple perspectives and ways of discussing a scenario. But hyperscenarios are potentially active documents that transcend the standard navigational interaction of much hyperdocumentation.

The following section of this paper describes the issues involved in designing a hypermedia scenario framework. We present a brief overview of **SGML** (Standard Generalized Markup Language) and why it was chosen as a basis for the scenario scripting language. A conceptual model of scenarios is illustrated using an example of the proposed SCML (Scenario Markup Language). The third part of the framework discussion looks at tool functionality and issues involving where capabilities should reside, in the language or in the applications. The third section of the paper gives an example application of a hyperscenario using an Army logistics problem.

DESIGNING A HYPERSCENARIO FRAMEWORK

Prior to constructing our hyperscenario environment, the decision had to be made concerning representation. Scenario information needed to be stored in a form that was general enough to describe the structure of any scenario, yet expressive enough to support different external views of the data. The internal representation or structure had to be separated from presentation style. The choice of a markup language scripting strategy supports both these objectives. Markup languages have been used for many years to support the creation of online and printed documentation.

There are two categories of tools used for preparing documentation: text formatters and markup languages. Text formatters (such as *NROFF*, *LaTex*) require the preparer to handle the details of a document's presentation. Very precise commands and macros are setup to handle margin placement, font size, font style, page numbering, etc. These commands control how the document is to be displayed and generate specific commands to control the printer. Markup languages are concerned with the *structure* of the document. These languages allow the user to specify the document's components, such as headings, paragraphs, tables and the like, without require knowledge of the type and style of output device. The strength of this approach is that the appearance of the document is left to the application tool/environment. For example, A paragraph heading level one may mean 12 point Roman font on one machine and 10 point courier on another. The disadvantage is that sometimes it becomes difficult if you want the appearance of a component to be the same on every device. Markup languages are used to "model" the document by defining what are the necessary parts. A model of a book would consists of chapters, paragraphs, and sentences. A markup model of a report might contain a table of contents, sections, figures, and an index among other things.

SGML is the meta-language that is used as the basis for most markup languages.[12] SGML-based languages all use a similar "tagged attibute" approach to modeling the documents that are being processed. A tagged attribute, such as `<TITLE>`, is used to denote the beginning of a document component. An ending tag, in this instance `</TITLE>`, marks the end of the occurence. It is between these tags where the text would be inserted.

Markup languages are defined from SGML by creating a **DTD** (Document Type Definition). It is the DTD that encodes the model of the document by assigning the allowable components and structures. A DTD can be used to generate an intermediate form of a document to

support translating from one format to another. Many desktop publishing applications, such as FrameMaker and Interleaf, use their own SGML-based DTD to create ascii versions of the documents. FrameMaker MIF files are an example of documents that use the FrameMaker DTD. It is this context-independent format that makes it easier to convert from Framemaker to other word processing tools. The most widely used SGML-based language is HTML (Hypertext Markup Language). HTML documents are the basis for the World-Wide Web, with the encoding of links (referred to as hyperlinks) between pages as tagged attributes.

Designing a DTD consists of determining the entities, elements, and attributes required to represent every possible combination of components within the document structure. The designer has to make decisions on component placement and allowable forms of the document. For example, in a DTD for a report, does each section have to have a title? Can a report have chapters instead of sections? A DTD is used by the application program to validate that a document is of the appropriate type. Figure 1 illustrates how elements are described in SGML syntax. There are 3 parts to each Element declarations: **Name, Minimization, and Content**.

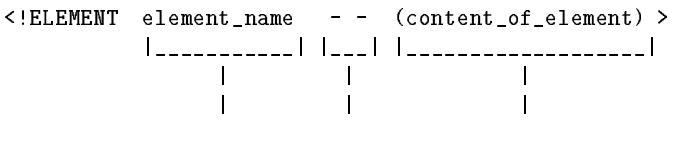

Figure 1: SGML Element Syntax

The **Name** is the label of the tagged element. A group of names in parentheses could appear, separated by either a comma, vertical bar, or ampersand. The **Minimization** explains whether there should be an start-tag and end-tag. A "-" means the tag is necessary, a "o" means the tag is ommitted. For example " - - " means the element has a start-tag and an end-tag. The content explains what each element can contain. **Contents** are either other elements or a reserved word. The reserved word **PCDATA** is used when the parsed data can be any valid character data.

DTD items are grouped within parentheses. The group connectors have the following meanings:

, (Comma) - means the items appear in this order.

| (Vertical Bar) - means that one of the items separated by the bar may appear.

& (Ampersand) - means that the ordering doesn't matter.

There are punctuation marks within the elements that have special meanings. These marks are occurrence indicators which show how many of the item may appear in the element.

? (Question mark) - means either one occurrence of the item exists or it isn't present.

* (Asterisk) - means the item is either absent or appears more than one time.

+ (plus) - means one or more of the item is present.

The hyperscenario environment that we are constructing will use a markup language called **SCML** (Scenario Markup Language). SCML has all the advantages that mentioned previously, most notably, easy conversion from one format to another. Like HTML, SCML can handle multimedia types of information, not just text. Many of the same design decisions that went into designing the DTD for HTML influence the choices for SCML. However, SCML is not an extension of HTML. The design of the SCML DTD is based on modelling a scenario as a document (or collection of documents). The relationship and occurrence of scenarios components are based partially on the scripting language used to represent the scenario and also on the application tools that are to interpret the language. In the following section, we describe the concept of a scenario and how it might be represented with a proposed SCML DTD. Describing the SCML DTD and scenario model at the same time will make it easier to see how we have modelled the scenario structure in the markup.

A Conceptual Model for Scenarios

The scenario model is derived from a story grammar-based view of information[14]. This structure uses the information from story grammars to describe system behaviors, context, and consequences. Story grammars can be considered as narratives or scripts that describe the actors, actions, and potential outcomes of a sequence of events[13]. The reason that story grammars are useful in describing scenarios is associated with the use-oriented nature of scenarios. Scenarios are used to make learned artifacts explicit, i.e. to present to the user a concrete instance of an activity. This implied story-telling structure of scenarios is analagous to a description of a movie or television program. Just like a play, useful scenarios have a setting, a background, and some narrative that describes what is to take place. Background would be existing documentation, specifications, and initial requirements for a system. The setting would present some description of where the task or activity is to take place. Remember, that the ultimate purpose of a scenario is to describe a specific task activity within a system. A narrative would be a list of steps or in-

structions for what sub-activities are to take place and perhaps some dependency or sequence information.

Figure 2 gives the scenario model represented as a DTD. Using the SGML element syntax described earlier, the components of a scenario and their interrelationships are encoded as rules. The structure reflects some design decisions based on how the scenarios could be used as well as structures that define what we are referring to as scenarios.

```
<!- <!DOCTYPE  scenario [ ->

<!-  PUBLIC IDENTIFIER of this DTD,
  "scenario.dtd (Scenario)"
->
<!- ************************************************************** ->
<!-   NAME:     Scenario DTD                                       ->
<!-                                                              ->
<!- ************************************************************** ->

<!ELEMENT  scenario - - ( title,purpose?,role+,narrative,outcome? ) +( anno-
tation )>
<!ELEMENT  purpose - - (#PCDATA) >
<!ELEMENT  ( role — actor — stakeholder — component ) - - ( name, respon-
sibility?, influence?, action? >
<!ELEMENT  narrative - -  ( title? , setting , ( script — sequence )  >
<!ELEMENT  outcome - - (#PCDATA)>
<!ELEMENT  title - - (#PCDATA) >
<!ELEMENT  script - -  ( description? , episode+ >
<!ELEMENT  sequence - -  ( description? , scene+ ) +( annotation )>
<!ELEMENT  description - - (#PCDATA) >
<!ELEMENT  episode - -  (description? , action+ , goal ) +( annotation )>
<!ELEMENT  scene - - ( description? , setup , goal ) +( annotation ) >
<!ELEMENT  ( viewpoint — setup ) - - ( cut+ ) >
<!ELEMENT goal - - ( description? , obstacle+ & condition & constraint* & op-
erationalization* >
<!ELEMENT  operationalization - - ( action+ ) >
<!ELEMENT  ( condition — constraint ) - - (#PCDATA) >
<!ELEMENT  obstacle - - ( description? , mitigation* & defense* ) >
<!ELEMENT  cut - - (#PCDATA) >
<!ELEMENT  action - - (#PCDATA) >
<!ELEMENT  setting  - - (#PCDATA) >
<!ELEMENT  mitigation - - (#PCDATA) >
<!ELEMENT  defense - - (#PCDATA) >
<!ELEMENT (annotation — question — response — assumption — comment ) -
- (#PCDATA) >
]>
```

Figure 2: DTD for a hyperscenario Document

A scenario always contains one or more episodes. These episodes are a list of actions that are grouped based on a particular outcome. In our DTD, a script is the collection of episodes for a scenario. The individual actions within an episode could vary. For example, in the case of a customer banking system, a *Withdraw-Money-From-Checking* episode could be accomplished by the use of an automated-teller machine or a face-to-face transaction with a teller. It is this variability of actions within an episode that allows for the examination of alternative design approaches within scenario analysis. Episodes have a stated goal in mind, a reason for the list of actions that occur. It is these goals that can be likened to a plot in a movie script. The usefulness of a scenario is it's purpose, i.e. what is the scenario attempting to illustrate?

The actions within an episode operationalize the goals for the system. For the sake of this model, the goal entity is not further divided into subtypes, although some approaches separate goals into achievement goals, maintenance goals, quality goals, etc.[1] Each of these goals may contain sub-goals that need to be accomplished, or may themselves be sub-goals. The overall purpose of an episode is to satisfy some goal or goals within the scenario. This satisfaction of goal requirements can be blocked by the presence of obstacles. An obstacle can impede the accomplishment of a goal. The successful completion of a goal requirement is established by the presence of one or more desired conditions within a system. According to syntax in Figure 2, a goal has one or more obstacles and a least one condition. There may or may not be constraints on the goals or ways to operationalize the goal. To accomplish a goal,the condition that established the necessity of the goal has to be met. It is these conditions that are blocked by obstacles. In some systems, determining the goals is preceded by determining the obstacles against successful completion of a task or activity. The goals are then introduced to either mitigate or defend against any obstacles.

A role, actor, stakeholder, or component are all different perspectives or views of the system being analyzed. They can be defined by a name, a responsibility, an influence, and actions that they perform. It is the responsibility of actors, for example, to perform the actions within the episodes. Stakeholders (some of which may be the same actors) are influenced by the results of those actions. each of these views could be automated or could represent particular users within the system. Roles are also responsible for the achievement of goals. A role illustrates the different viewpoints into the system. The designer, developer, technical writer, test and evaluation staff, and customer all have different viewpoints into the system, and ,therefore, potentially different stakes in the achievement of specific goals.

The result of scripting through the episodes within a particular scenario is that they lead to an outcome. It is this outcome that allows for the refinement of the requirements of a system. If the desired outcome is generated by applying several different scenarios to meet a specific goal, it is then necessary to determine how those outcomes influence the individual stakeholders for the potential system. It is the communication that is derived from an evaluation of the outcome of a scenario that is useful to not only determine the salient scenarios, but also to refine the goals of the system to more closely match user needs. This discussion takes the form of annotations on the scenarios that are a list of raised questions and proposed responses to an outcome from a scenario.

This DTD is a proposed template for SCML for scripting our hyperscenarios. Some of the details in constructing this language was based on the model of how a scenario fits together. What are some of the other design decisions that have solutions to be reflected in SCML? One problem is how to handle alternate internal representations of the same data. In order to construct scenarios that can be represented a myriad of ways, the raw data for those views have to either be directly present in the documents or the mechanisms for generating them have to be there. One way of accomplishing this is having tagged attributes with alternate values dependent

upon a predefined mode. For example, if the applications viewer is constructing a storyboard version of the document, those components that have a GIF representation would be processed. There would be a default GIF image for those components that do not have the data stored in that manner.

Another design problem is document access and navigation. How will the relationship between hyperscenarios documents be maintained? The SCML approach to this problem will use attributes that have a revision numbering scheme similar to the indices generated by revision control systems like **CVS** (Concurrent Versioning System)[3]. CVS uses a hierarchical numbering system to keep track of multiple document versions within a repository. CVS is used in collaborative software development tasks where multiple users will be sharing and modifying a collection of documents. The numbering strategy is also designed to support alternate branching constructs.

Hyperscenario Tools: What are the capabilities

The application tools that interpret the hyperscenario documents are the other half of the creation of the framework. Similar to Web Browsers (such as Netscape or Internet Explorer), it is the job of the application tool to define the appearance of the hyperscenario pages. Although there could be a suite of tools to manipulate scenario information, for the purpose of this discussion, we will describe a single tool that exhibits certain functions. This tool, which we'll call **SCMLView**, will have the SCML DTD encoded into it. This encoding will support hyperscenario authoring, through context-sensitive help.

One of the most straightforward tasks for the hyperscenario tool is translating the scenario data to different forms, depending upon user requirements. Creating a storyboard version of a scenario would be useful in decision-making processes. You can think of this arrangement of scenario episodes as similar to the slide sorter function that is available in PowerPoint presentation software. The difference here is that each image represents an event that occurs as part of the scenario. The connections between the events is not arbitrary, in that only certain arrangements of images would make sense. A storyboarding process would involve adding events, repositioning events, even removing events from the scenario. Each of these changes would be tracked through the tool by an external control structure.

There is also the function of creating standard documents from the hyperscenario. The type of documents that could be generated would be instructional, such as user manuals. Having the underlying structure for the user manual exist as a scenario is userful for troubleshooting. Since scenarios are specific cases of a system behavior, the user manual would contain linkages to example system problems and troubleshooting suggestions.

Just as there were specific design decisions to address for the SCML portion of the framework, we need to look at some of the issues involved with the application tool. One capability that needs to reside in SCMLView is a session management function. There has to be some way of keeping track of the state of the hyperscenario, during and between viewing/analysis sessions. This is even more difficult to manage when you allow for multiple users with potential for extending the hyperscenario. This can be handled using a client-server approach to the hyperscenarios. Each time an SCMLView client is launched, an external lookup table will be setup to keep track of changes to the hyperscenario.

Another capability within SCMLView is multiple entry or exit points from the scenario. It should be possible to analyze the scenario from any point, not just at the beginning. Traditional HTML pages suffer from the fact that although you may enter a collection of pages at any point (assuming you have the proper url), you cannot move to previous pages unless you have already visited them or there is an explicit link to go back. There is no "awareness" within HTML pages about the surrounding documents.

Much of the complex functionality within SCMLView will require the control structure that we keep referring to. This structure is similar to a Makefile on a Unix System. A makefile is used to compile a program in the most context aware manner. The makefile encodes rule about the program being compiled and its environment. We not only need to know about the pages with the hyperscenario, but also when and how those pages are being accessed. Collaborative sharing of a hyperscenario requires this functionality the most. Again, SCMLView will use a CVS-style method for controlling access to the document by mutiple groups of people. A repository of available hyperscenario pages must be accessible to SCMLView without the added overhead of actually maintaining a repository. Some of the state of the individual pages is embedded in the language, such as revision date, revision number, branching construct, etc. The combination of scripting support for concurrency and managing simultaneous access within SCMLView will handle multi-agency problems.

If these scenario documents are used for collaborative discussions, there would need to be support for both synchronous and asychronous interaction. The synchronous interaction would not have to happen completely in real-time. There are existing tools that support group-aware shared resources (such as liveboards) that could be tied in to the documents in the appropriate places. It should also be possible to separate into subgroups during a collaborative session, sort of the "cocktail party" style of interaction where pieces of the same scenario are being discussed by separate groups.

There are some other functional capabilities that could be built into SCMLView:

Directed Views-Immersion in the Scenario. It would be useful during a development process to be able to take

on a different perspective for the system, even the perspective of the system itself. Essential, the participant could chose to look at the same scenario while assuming different roles.

Alternate Path Analysis. Scenarios are at there most useful when they are the focus of "What-if" studies. Being able to perform concurrent development or examine the impact of alternate paths will further enhance the completeness of a system review.

Levels of Abstraction - Zooming In/Out. Scenario discussions are notabe for the diverse backgrounds of the participants in the discussions. End users, developers, programmers, technical writers, and administrative staff can view the same scenario to get an understanding of a system or process. The level of abstraction, in other words the detail of the scenario, should be modifiable based on the user. Management may only want a high-level view of system. Technical writers may require a mid-level view that expresses enough of the interaction to describe what is going on in the system or process being discussed. The developer of the system may want a very detailed, lower-level representation of the scenario for test and evaluation purposes.

Dependency Analysis. What are the impacts of certain changes in a proposed system? What components are responsible for performing a specific action in the system? Is there a required sequence of events that have to occur in order for this scenario to proceed to the desired outcome. Because there are language constructs within SCML that establish relationships between components, an application tool could analyze the dependencies in the scenario. These dependencies would help prevent modification to a scenario that would introduce instability into a system, given its current state.

Online Tutorial. Many companies and government agencies attempt to build an organization process model of how the business works. This process model is supposed to incorporate the mission statement, business rules (both explicit and implied), amd company policy. By creating scenarios based on the identified processes, information flows, constraints, and policies, hyperscenarios could be constructed for training purposes. For example, let's say an organization has a travel preparation process for attending out-of-town conferences. This travel process involves interacting with several tools and communicating with different administration personnel. A travel preparation scenario could include film clips representing certain activities to be accomplished, it could spawn tools to handle travel document preparation, or even prepare fax documents for hotel reservations. A new user would simultaneously understand the structure of the organization and gain assistance in performing a specific task.

APPLICATION OF A HYPERMEDIA SCENARIO: AN ARMY LOGISTICS PROBLEM

We have discussed some of the design issues involved in constructing a hyperscenario framework. Some of the control structures are reflected in SCML and some in the application tool. In this section, we present a specific example of how the hyperscenario framework will support several cognitive tasks. For the sake of discussion, we will refer to the proposed application tool SCMLView that we described earlier.

To illustrate the utility of defining scenario documents using our proposed DTD, we have chosen a small scenario from the Army. This scenario appeared as a practical exercise in Army doctrine during an government training course at the Army Management Staff College. The situation described in the exercise did not actually occur, but it is indicative of the normal mission description that are presented to installation commanders. The mission was described as follows:

The Division Commander of the 15th Infantry Division has just been informed that the division Military Police Company has been tasked to deploy to Panama. This deployment is a result of a rotational requirement which sends TOE[1] Military Police Companies from CONUS[2] to augment existing Military Police assets in Panama on a quarterly basis. This unit is scheduled for deployment in four months. It will remain in Panama for three months and then return to home station. While in Panama it will be assigned to a Joint Task Force with a continuing mission of providing support to the Panamanian government.[2]

The practical exercise included other information such as objectives of the Joint Task Force and specific missions for the Military Police (MP) company. A detailed scenario containing all of the information from this particular mission would be a collection of scenario pages, each representing a sub-scenario of the overall mission.

[1] Tables of Organization and Equipment - A document that lists full strength requirement for a unit, including required equipment.

[2] Continental United States

```
<scenario>
<title>Military Police Unit Deployment</title>
</purpose>Describe the logistics of a rotational deployment of the Division Military Police (MP) unit</purpose>
<Role>
<LI><stakeholder>Division Commander</stakeholder>
<LI><actor><stakeholder>MP Unit Company Commander</stakeholder></actor>
<LI><actor>1st Platoon Leader</actor>
<LI><actor>2nd Platoon Leader</actor>
<LI><component>MP Company</component>
</Role>
<narrative><a href="background.txt">Requirement for MP Unit deployment</a>
<annotation>This deployment occurs on a quarterly basis IAW DA directives</annotation>
<script>
<episode><description>Deployability Status of Personnel</description>
<goal>Determine if MP unit at TOE levels for deployment
<obstacle>Current disposition of all personnel unavailable
<mitigation>Assume TOE levels of personnel</mitigation>
<defense>Maintain current database of active/inactive personnel</defense></obstacle>
<condition>MP Unit deployable if at TOE levels<\condition>
<constraint>Unit not deployable if adequate transportation not available<\constraint>
<operationalization>
<action initiator=MP Unit Company Commander>Request briefing from platoon leaders and support staff on deplo
status of MP unit<\action>
<action initiiator=1st Platoon Leader>Issue recall of all active and non-active persoonnel not on administrative or
sick leave<\action>
<\operationalization><\goal><\episode>
   Additional Episodes are listed here
<\script>
<\scenario>
```

Figure 3: Army Logistics Scenario Document

Figure 3 shows a possible encoding of the exercise scenario using SCML. To keep the example brief, we are only focusing on one aspect of the scenario, that of determining the status of the unit being deployed.

The first thing to notice is the statement of the scenario's purpose. It is this purpose that will be required in order to maintain a thread of related scenarios throughout the document walkthrough. Additional attributes on the `<purpose>` tags as well as capabilities within the application tool would be required to establish the relationship between documents based on the stated purpose. The scenarios could be approached hierarchically, as in the zoom in/out featured that was discussed earlier. In that instance, the document and the application tool would have to be able to perform some decomposition of purpose in order to associate relevant documents in the collection of pages.

The next block of the example presents an itemized lists of roles, that are bracketed by the appropriate tag. Each entity in a scenario could maintain different roles in the system, depending upon the view of the scenario and the activity being analyzed. This is reflected in the fact that an MP Unit Company Commander can be considered a stakeholder and an actor within the system. Which role is active is determined by the position in the scenario walkthrough.

It should be possible to include external documentation when discussion a scenario. This documentation would become part of the narrative that explains background information on the scenario. In our example, we've used an HTML tagged anchor to generate a hyperlink to an external document. This document could contain policy and guidance information for unit deployment or other administrative assistance. The annotation tag is important in that this is a mechanized for presenting assumptions or open issues about the scenario under review. To use the screenplay metaphor, the annotations could be considered as liner notes or stage directions written onto a script. These notes are helpful in allowing the actors to interpret how the director wants a particular scene accomplished, and in the case of our deployment mission, it establishes where the orders for this mission originated.

The script tags are used to bracket a list of episodes that are the main portion of a scenario. Each episode contains a description of what the episode is for and its goal. Notice that the goal entity contains any obstacles, mitigation strategies, obstacle defense, conditions, and constraints. This makes sense because the other entities presuppose the existence of a goal and every episode has a goal. There may or may not be an obstacle to a goal, but likewise, if there is a obstacle there should probably be some mitigation or defensive strategies assigned to it. Now part of the reason for a scenario walkthrough may be to determine these obstacles and mitigation strategies, so they don't have to exist for the document to be a complete scenario.

The actions that operationalize a goal are also bracketed within the tags. This is one way the document structure establishes a relationship between the goals and the actions that support those goals. If the goal and action relationship are defined properly, they could be reused throughout the scenario given the appropriate entry and exit points. For each action defined, there is an initiator attribute that assigns responsibility for the action. Each initiator named must be consistent with a role that was designated at the beginning of the scenario document.

This example scenario is at a fairly high-level in that particular details of the deployment are left out. Some of the information necessary to augment the scenario would be automatically inserted by a domain-sensitive hyperscenario tool. For example, the TOE for an Army Unit (the number of required personnel and resources) are standard based on the type of unit. This information could be retrieved from an online database of Army policy and procedures documents. Most of the scenario details would be the result of guided scenario authoring in SCMLView. The initial version of the Army Logistics problem could consist of one hyperscenario document that encodes this level of information. As the scenario is discussed, sketchy details would get filled in, creating other hyperscenario documents. The management and manipulation of the collection of documents is also one of the tasks of SCMLView.

SCMLView would guide the translation of this textual description into a hyperscenario. A series of dialogue boxes with specific questions relating to scenario compo-

nents would appear. There would be default questions that are required in order to format the scenario structure. The questions would concern episodes or events within the scenario, actors, actor responsibility, goals for each event, and sequence of events. The user would have the capability of adding domain-specific questions that could be used to insert appropriate information. Any Army situation involves an implied chain-of-command or ranking structure that establishes authority for carrying out mission objections. A domain-specific question about chain-of-command could get mapped to scenario stakeholders, actors, and actor responsibilities (actions).

The information encoding in SCML is also done through SCMLView. The default textual representation of the scenario would be written into the SCML with the appropriate hyperlinks establishing episode sequence. If there are other formats for the scenario attributes, these styles would get stored as alternate representations of the information that would be accessed based on the modality of the SCMLView tool. For example, the Army has standard symbols or icons to represent units. A tank division has a completely different symbol from an infantry division. These icons are stored as alternate display types associated with the same system actor. A labeled icon for the 15th Infantry Division would be in the SCML code along with the textual description.

To illustrate how the encoded Army Logistics hyperscenario would be used, the next three sections are themselves scenarios about the scenario.

Commander's Intent

In any military situation that involves a mission statement, there is always the problem of commander's intent. Commander's intent refers to deriving the exact meaning and method of accomplishing the goal as it is implied in the orders. The hardcopy orders are brief and to-the-point, relying on the subordinate officer's training in Army doctrine and policy to fill in the particulars. This flexibility is somewhat necessary in potential combat situations that are dynamic and volatile. But the problem is that sometimes the commander had a specific approach to the tactics in mind that were not explicit in the orders. A recent study performed by cognitive researchers at West Point showed that subordinate officers would sometimes carry out orders in ways that did not follow the commander's desired method for reaching the objective.

Let us suppose that our Infantry division commander wishes to take advantage of our hyperscenario tool and decides to encode his orders as a scenario. He records an audiovisual message that outlines the tasks and a particular way of accomplishing them. The audio version of the recording is transcribe into a textual description. As the video stream is converted into a Quicktime video stream, it is segmented into episodes based on the events being discussed. In some cases, it may not be possible to separate the video streams into legible segments. In those instances, a hyperscenario document containing the entire video gets associated with the textual narrative.

The Military Police company commander receives the orders online. If he wishes to see a standard order document, the scenario information can be mapped directly onto the convential orders and displayed. If the commander wants to have more detail than is reflected in the order, a detailed narrative from the transcribed recording is supplied. The audio portion or the entire Quicktime movie (depending on available bandwidth) can be reviewed to pick up body language or verbal cues to any implied method of meeting the mission objective.

Tactical Planning

In the commander's intent problem above, one issue is there may be many ways to accomplish the same objective, even within the auspices of Army doctrine. The Military police company commander needs to determine the best schedule for deploying his troops to Panama to support the Joint Task Force while maintaining the appropriate level of readiness at the installation. He decides to have a collaborative meeting with his executive officer and platoon leaders. Because they need to be able to determine dependency and perform alternative path (choice) analysis, he decides to use SCMLView in the storyboarding mode. In this configuration, the tool examines the hyperscenario and retrieves the icons representing the different actors and stakeholders in the system. Constraints, obstacles, and goals to the particular scenario are associated with each event on the storyboard. As the decision group manipulates the icons of the storyboard, the dependency information in the hyperscenario gets updated. Actually, the manipulation of the hyperscenario is being stored as a alternate collection of hyperdocuments. By having the hyperscenario dependency information change based on storyboard configuration, it would be possible to tell what are viable options for accomplish the objective. For example, there may be a mitigation strategy in the original scenario that handles what to do if there is not a ship available for transporting the troops to Panama. This strategy may only work if the deployment is done with a certain date range when a ship of the appropriate size would be available. If one of the alternate scenarios would delayed the deployment past the acceptable date range, this would be indicated in the storyboard as a potential conflict.

After-Action Reports

It is eight months later and the Military Police company has rotated back to the installation after finishing the deployment. The division commander wants a report on how well the mission progressed and any lessons learned for future deployment. During the deployment, the company commander had all incident reports encoded as scenarios using the SCMLView tool. Also, any collaboration with the Joint Task Force or the Panamanian Defense Forces was noted and translated into scenario form. The company commander was able to prepare overview reports on exactly how resources and personnel were used during the deployment by directing

SCMLView to summarize the information under each major category of tasks. These summaries could be briefed to the division commander to show how successful the deployment was in meeting mission objectives. Any problem scenarios that came up could be reviewed to create mitigating and defense strategies for furture deployments.

DISCUSSION

The idea for this research into hyperscenarios came as a result of looking at the issues involved in handling a scenario as a document. Since scenarios are stories about a system, what metaphor can we tap into for insight about describing those stories. The concept of a screenplay from the film industry was a good example of what we wanted to accomplish with scenarios as documents. Some of the terminology used in the film industry is part of the DTD for SCML.[7] Just like scenarios, a screenplay is a story about one particular situation. The movie *Titanic* is one interpretation of its screenplay and is an instance of behavior in the action-adventure movie genre. There could be several ways to represent the same screenplay: a storyboard of scenes, the soundtrack, the advertising trailers, etc. But in all cases, the structure of the screenplay document remains consistent. The content of a screenplay can vary significantly. A screenplay can describe movies in different genres, such as horror, comedy, drama, and the like. A screenplay is also used to describe the activities in television productions. The commonality among all screenplays is the narrative structure. There is a standard way of specifying actors, settings, scenes, and actions. The analogous document in theatre or radio productions is a script. Basically, the same concept holds. The salient parts of the documents are understood by the industry and relatively brief instructions can convey large amounts of information. As SCML gets further refined, there may be additional notions from the theater industry that will become incorporated into the language.

In her book *Computers as Theatre*, Laurel talks of the need for new design approaches for human-computer activity.[9] Most of the human-computer interaction revolves around the user interface. Direct-manipulation systems that use the desktop metaphor allow users to view the computer as a collection of objects analagous to real world objects (files, folders, etc.). Laurel presents the notion of *direct engagement* systems that enable users to work directly in the activity of choice. This can be done by manipulating symbolic tools or immersion in a virtual environment. Although the hyperscenario framework isn't being designed as a virtual reality environment, many of the SCML structures could support the type of access that would qualify as direct engagement.

Scenarios are an organizing principle for simulation-based and role-playing games. CD-Rom video games, such as *DOOM*, *Tomb Raider*, and *Myst* allow a player to immerse themselves into a scenario where the sequence of events are based upon player action. At any given point within an adventure game, all possible scenarios of action have been pre-determined by the games designer. There may be some leeway in how the player accomplishes a particular task, but for the most part the system can only respond in limited ways. Some of the most complex interaction in adventure games use the same method as MUDs/ MOOs (Multi-user Dimension/MUD Object-oriented). In the Early 80's, text-based MUDs such as the *Hitchhikers guide to the Galaxy* allowed for a large number of outcomes based on a handful of potential moves. MOO environments like *SimCity* create virtual worlds where users can collaborate on design and share ideas by constructing scenarios. The **MediaMOO** project at MIT's Media Lab is a text-based, networked, virtual reality environment that was created to help media researchers. The "constructionist" philosophy of this environment is that people learn when that build personally meaningful artifacts.[4] The participants have flexibility in what they wish to create, constrained only by the objects that are present in the system to manipulate. Scenarios are useful methods for building virtually communities on MUDs/MOOs.

Scenarios are related to the concepts of actemes and episodes introduced by Rosenberg in *The Structure of Hypertext Activity*.[15] An **acteme** is a low-level activity, such a following a link. A collection of actemes are termed an **episode**. These actemes relate to the actions of our hyperscenario in that they are discrete events that occur, in this case, navigation between hyperlinks. The episodes are also a purposeful collection of the events in some sequence that has meaning to the user. The difference is that the structure of the language associates the actions with the episode. What constitutes an action depends upon the overall purpose and presentation form of the scenario, however, the episode-action relationship is what is being maintained. If link following are the actions for a particular scenario, then the valid path linkages (and navigation style) are decided ahead of time.

The purpose of this paper was to describe research in defining a hypermedia scenario framework. Scenarios are stories of system behavior that are used for comprehension, decision-making, and design. These scenarios can be represented as documents with interrelationships betweens the components in the document structure. Choosing a SGML-based approach to scripting these documents allows the language to separate internal format from presentation style. The hyperscenarios defined from this markup language, which we refer to as SCML, can contain textual, graphics, and audiovisual scenario information. An appropriate set of tools to interpret the SCML code support the creation of different representations of scenarios (such as storyboards, audiovisual segments, narratives, etc.) from the same data. Interactions between SCML and tool functionality will also support scenario authoring, dependency analysis, and structured walkthroughs.

REFERENCES

1. Annie I. Anton. Goal-Based Requirements Analysis. In *Proceedings of the Second IEEE International Conference on Requirements Engineering*, April 1996.

2. Army Management Staff College, Fort Belvoir, VA, USA. *Strategy, Forces, and Doctrine: Practical Exercise*, January 1995.

3. Brian Berliner. CVS II: Parallelizing Software Development. White paper, Prisma, Inc., Colorado Springs, CO, 1990.

4. Amy S. Bruckman. The MediaMOO Project: Constuctionism and Professional Community. *Convergence*, 1(1), 1995.

5. John M. Carroll, editor. *Scenario-Based Design: Envisioning Work and Technology in System Development.* John Wiley & Sons, Inc., New York, 1995.

6. John M. Carroll. The Scenario Perspective on System Development. In *Scenario-Based Design: Envisioning Work and Technology in System Development* [5], pages 1–15.

7. Edward Dmytryk. *On Film Editing.* Focal Press, Boston, 1984.

8. Morten Kyng. Creating contexts for design. In Carroll [5], chapter 4, pages 85–107. Aarhus University, Department of Computer Science.

9. Brenda Laurel. *Computers as Theatre.* Addison-Wesley Publishing Co., Reading, MA, 1991. QA 76.9 H85 L38.

10. Robert L. Mack. Discussion: Scenarios as Engines of Design. In Carroll [5], chapter 14, pages 361–386. IBM T. J. Watson Research Center.

11. Bonnie A. Nardi. Some Reflections on Scenarios. In Carroll [5], chapter 15, pages 387–399. Apple Computer, Advanced Technology Group.

12. National Institute for Standards and Technology, Gaithersburg, MD, USA. *Standard Generalized Markup Language (SGML)*, September 1988.

13. Colin Potts. Requirements Completeness, Enterprise Goals and Scenarios. Research Report, August 1994.

14. Colin Potts. Using Schematic Scenarios to Understand User Needs. In Gary M. Olson and Sue Schuon, editors, *Proceedings of the Symposium on Designing Interactive Systems: Processes, Practices, Methods and Techniques*, pages 247–256, New York, 23–25 August 1995. ACM Press.

15. Jim Rosenberg. The structure of hypertext activity. In *Proceedings of the 7th ACM Conference on Hypertext*, pages 22–30, New York, 16–20 March 1996. ACM Press.

16. Roger C. Schank. *Tell Me A Story: Narrative and Intelligence.* Norwestern University Press, Evanston, Ill, 1995.